Base Path Testing

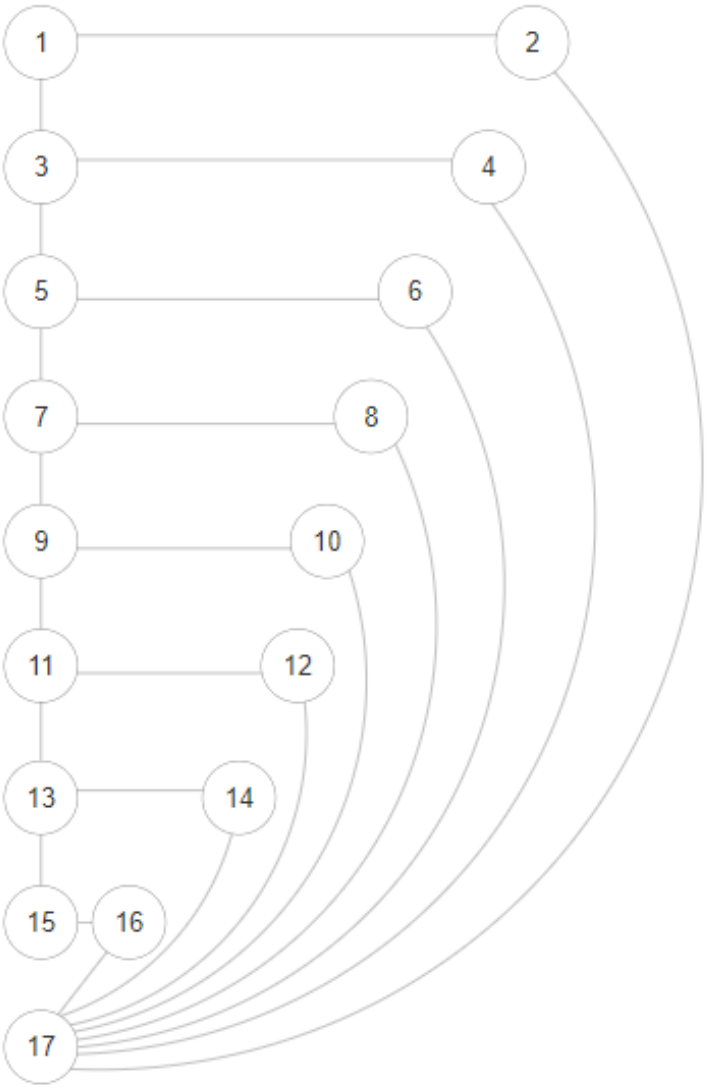
Base Path

Finding the paths is important to evaluate which are more common and which are longer. The shortest paths should be the most common, and the longest the least common. This is to optimise the performance of the program and reduce the risk of crashes while running.

To do this, we must highlight the possible paths the code can take. The first step in this process is evaluating where paths can diverge using a flow graph, as shown below.

```
    private function setBadge($numBookings){
1-      if($numBookings <= 5){
2-          $this->badge = "Wooden";
        }
3-      else if($numBookings <= 10){
4-          $this->badge = "Stone";
        }
5-      else if($numBookings <= 15){
6-          $this->badge = "Iron";
        }
7-      else if($numBookings <= 20){
8-          $this->badge = "Bronze";
        }
9-      else if($numBookings <= 25){
10-         $this->badge = "Silver";
        }
11-     else if($numBookings <= 30){
12-         $this->badge = "Gold";
        }
13-     else if($numBookings <= 35){
14-         $this->badge = "Platinum";
        }
15-     else{
16-         $this->badge = "Diamond";
        }
17- }
```

Flow Graph



Now that we have a flow graph of the code, the next step is to find out how many paths there are.

Cyclomatic complexity

Cyclomatic complexity is simply how complex a cycle is, or in our case, how many paths there are.

It can be calculated with the following formula.

V(G) = E – N + 2 where E = edges and N = Nodes

In this base flow graph, there are 23 edges and 17 nodes.

V(G) = 23 – 17 + 2

V(G) = 8

Now that we have the number of paths, we can evaluate what the paths are, ensuring no paths are repeated or missed.

Paths

The following paths were found.

Path 1: 1 – 2 – 17

Path 2: 1 – 3 – 4 – 17

Path 3: 1 – 3 – 5 – 6 – 17

Path 4: 1 – 3 – 5 – 7 – 8 - 17

Path 5: 1 – 3 – 5 – 7 – 9 – 10 – 17

Path 6: 1 – 3 – 5 – 7 – 9 – 11 – 12 – 17

Path 7: 1 – 3 – 5 – 7 – 9 – 11 – 13 – 14 – 17

Path 8: 1 – 3 – 5 – 7 – 9 – 11 – 13 – 15 – 16 – 17

These paths show that the paths with the least booked lessons are shorter, with the number of lessons booked increasing with the path length. As less members are likely to reach higher numbers of lessons booked, these paths are optimal of our applications performance.

Equivalence Partition

To ensure the paths referred to above produce the correct results, we conducted an equivalence partition test on them.

This involves testing each path with several values, one at the lower and upper bounds of the path, one just beyond the lower and upper bounds of the path and one in the middle of bounds of the path.

The following results were found.

E Classes tested:

Lower then bounds <in bounds> higher then bounds.

Path	NumBooking Range	E Classes Tested	Expected Result	Actual Result
1	0 – 5	0, 3, 5>, 6	Wood	Wood
2	6 – 10	5, <6, 8, 10>, 11	Stone	Stone
3	11 – 15	10, <11, 13, 15>, 16	Iron	Iron

4	16 – 20	15, <16, 18, 20>, 21	Bronze	Bronze
5	21 – 25	20, <21, 23, 25>, 26	Silver	Silver
6	26 – 30	25, <26, 28, 30>, 31	Gold	Gold
7	31 – 35	30, <31, 33, 35>, 36	Platinum	Platinum
8	Greater than 35	35, <36, 10000	Diamond	Diamond